In this post, we are going to understand how to build a secure user authentication system using JSON web tokens (JWT) in Angular application.

To build the JWT user auth system, we will use the RESTful API and we will use node, express and mongoDB services.

We will be using the token-based user authentication RESTful APIs which we covered in our earlier tutorial.

**What is JWT (JSON Web Token)?**

JWT refers to JSON Web token. It’s a token in string form validated and generated by a web server. This string-based token helps in communicating between the client and the server.

Let’s understand how does the JWT help in transferring the data securely between the client and the server.

User information is sent to the client like username and password using an HTTP POST request to the webserver.

Web server identifies the user information and creates a token and send it back to the client. Client store that token into local storage or a session and also set it to the header. On the next HTTP call, that token is verified by the server, and web server returns the response to the client.

**Angular 13 JWT User Authentication Example**

* User can sign-in
* User can register or sign-up
* Store user data on MongoDB database.
* Creating Angular service to handle user auth API.
* Hide certain menu item based on authentication state
* Restrict user to access user profile page when the user is not logged-in.
* Store JWT token in local storage to manage the user session in Angular
* Store password in mongoDB Database using the password hash method with bcryptjs.
* Generate a JSON web token when a user logs in and store that token in a authorization header using HttpInterceptor class.

**Set up Angular Project**

Let’s get started by installing the basic Angular app, enter the following command in your terminal:

ng new angular-meanstack-authentication

Bash

Next, head over to the Angular user authentication project folder:

cd angular-meanstack-authentication

Bash

To make things simpler create a separate front-end and backend (server) in Angular app. Our Angular user auth app will have **signin**, **signup** and **user-profile** pages.

Create specific **components** folder in **src/app/components** in Angular app and create the following components in it.

ng g c components/signin

ng g c components/signup

ng g c components/user-profile

Bash

Next, install Bootstrap 4.

npm install bootstrap

TypeScript

Add the Bootstrap 4 stylesheet path in **angular.json** file.

"styles": [

"node\_modules/bootstrap/dist/css/bootstrap.min.css",

"src/styles.scss"

]

TypeScript

Start your Angular app.

ng serve --open

Bash

**Clone Node Js Token-Based Auth REST API from GitHub**

Run the following command to clone **Node.js token-based auth REST API** from GitHub in your Angular’s root folder.

git clone https://github.com/SinghDigamber/node-token-based-authentication.git

Bash

Get inside the server folder:

cd node-token-based-authentication

Bash

Next, install npm packages to set up and start the Node server:

**Run**`npm install`**to install required packages**

**Open other terminal run**`mongod`

Next, install the nodemon NPM package for development purpose with --save-dev attribute, it helps in starting the node server when any change occurs in the server files.

npm install nodemon --save-dev

Bash

Open terminal run `nodemon`

Following auth APIs we are going to use in this tutorial.

| **API Methods** | **API URL** |
| --- | --- |
| **GET (Users List)** | /api |
| **POST (Sign in)** | /api/signin |
| **POST (Sign up)** | /api/register-user |
| **GET (User Profile)** | /api/user-profile/id |
| **PUT (Update User)** | /api/update-user/id |
| **DELETE (Delete User)** | /api/delete-user/id |

Open API URL on <http://localhost:4000/api>

**Setting up HttpClient**

To handle REST APIs via HTTP requests in our Angular user authentication app. We need to import [Angular HttpClient service](https://www.positronx.io/angular-8-httpclient-http-tutorial-build-consume-restful-api/) in the auth module.

Import HttpClientModule service in app.module.ts file.

import { HttpClientModule } from '@angular/common/http';

@NgModule({

imports: [

HttpClientModule

]

})

Bash

**Creating User Authentication Service in Angular**

Now create Angular auth service and user class, these files will handle all the **JWT user authentication** related APIs in our project.

Inside the shared folder create shared/user.ts file and include the following code inside of it.

export class User {

\_id!: String;

name!: String;

email!: String;

password!: String;

}

JavaScript

Next, run below command to create user auth service.

ng g s shared/auth

Bash

Add the following code in the **shared/auth.service.ts** file.

import { Injectable } from '@angular/core';

import { User } from './user';

import { Observable, throwError } from 'rxjs';

import { catchError, map } from 'rxjs/operators';

import {

HttpClient,

HttpHeaders,

HttpErrorResponse,

} from '@angular/common/http';

import { Router } from '@angular/router';

@Injectable({

providedIn: 'root',

})

export class AuthService {

endpoint: string = 'http://localhost:4000/api';

headers = new HttpHeaders().set('Content-Type', 'application/json');

currentUser = {};

constructor(private http: HttpClient, public router: Router) {}

// Sign-up

signUp(user: User): Observable<any> {

let api = `${this.endpoint}/register-user`;

return this.http.post(api, user).pipe(catchError(this.handleError));

}

// Sign-in

signIn(user: User) {

return this.http

.post<any>(`${this.endpoint}/signin`, user)

.subscribe((res: any) => {

localStorage.setItem('access\_token', res.token);

this.getUserProfile(res.\_id).subscribe((res) => {

this.currentUser = res;

this.router.navigate(['user-profile/' + res.msg.\_id]);

});

});

}

getToken() {

return localStorage.getItem('access\_token');

}

get isLoggedIn(): boolean {

let authToken = localStorage.getItem('access\_token');

return authToken !== null ? true : false;

}

doLogout() {

let removeToken = localStorage.removeItem('access\_token');

if (removeToken == null) {

this.router.navigate(['log-in']);

}

}

// User profile

getUserProfile(id: any): Observable<any> {

let api = `${this.endpoint}/user-profile/${id}`;

return this.http.get(api, { headers: this.headers }).pipe(

map((res) => {

return res || {};

}),

catchError(this.handleError)

);

}

// Error

handleError(error: HttpErrorResponse) {

let msg = '';

if (error.error instanceof ErrorEvent) {

// client-side error

msg = error.error.message;

} else {

// server-side error

msg = `Error Code: ${error.status}\nMessage: ${error.message}`;

}

return throwError(msg);

}

}

JavaScript

* The **signUp()** method stores the user name, email and password in mongoDB database.
* By taking the help of bcryptjs, we are storing the password securely in the database.
* The **signin()** method allows the user to access in the app using JSON web token generated by node server.
* We are getting JWT token from the API response and storing in the local storage, then in the **getToken()** method, we are accessing the token via local storage **getItem()** method.
* The **isLoggedIn** method returns true if the user is logged in else returns false.

**Set JWT Token with Angular HttpInterceptor**

In this part of the tutorial, we are going to set the JSON web token in the header using Angular HttpInterceptor. To set the authorization header, first create the **authconfig.interceptor.ts** file in the shared folder.

import { Injectable } from "@angular/core";

import { HttpInterceptor, HttpRequest, HttpHandler } from "@angular/common/http";

import { AuthService } from "./auth.service";

@Injectable()

export class AuthInterceptor implements HttpInterceptor {

constructor(private authService: AuthService) { }

intercept(req: HttpRequest<any>, next: HttpHandler) {

const authToken = this.authService.getToken();

req = req.clone({

setHeaders: {

Authorization: "Bearer " + authToken

}

});

return next.handle(req);

}

}

TypeScript

Import the AuthService in and inject inside the constructor. In the **intercept(){…}** method call the **getToken()** method to get the JWT token then within the req.clone method set the Authorization header and call teh next.handle() method.

import { HttpClientModule, HTTP\_INTERCEPTORS } from '@angular/common/http';

import { AuthInterceptor } from './shared/authconfig.interceptor';

@NgModule({

declarations: [...],

imports: [HttpClientModule],

providers: [

{

provide: HTTP\_INTERCEPTORS,

useClass: AuthInterceptor,

multi: true

}

],

bootstrap: [...]

})

export class AppModule { }

TypeScript

Next, import the **HTTP\_INTERCEPTORS** in the **app.module.ts** file and set the HTTP\_INTERCEPTORS along with **AuthInterceptor** in providers:[...] array.

**Protect Routes with CanActivate**

Run following command to set up [CanActivate interface class](https://www.positronx.io/protect-angular-2-routes-with-canactivate-guard-for-firebase-users/), It stops visitors to access certain urls in the Angular app. In our case we only want logged-in users to access the /user-profile URL.

ng g guard shared/auth

Bash

Next, add the following code in the **auth.guard.ts** file.

import { Injectable } from '@angular/core';

import { ActivatedRouteSnapshot, RouterStateSnapshot,

UrlTree, CanActivate, Router } from '@angular/router';

import { Observable } from 'rxjs';

import { AuthService } from './../shared/auth.service';

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanActivate {

constructor(

public authService: AuthService,

public router: Router

) { }

canActivate(

next: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {

if (this.authService.isLoggedIn !== true) {

window.alert("Access not allowed!");

this.router.navigate(['log-in'])

}

return true;

}

}

TypeScript

Then, go to **app-routing.module.ts** file and import the AuthGuard interface class and inject the AuthGuard in the route as given below.

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { SigninComponent } from './components/signin/signin.component';

import { SignupComponent } from './components/signup/signup.component';

import { UserProfileComponent } from './components/user-profile/user-profile.component';

import { AuthGuard } from "./shared/auth.guard";

const routes: Routes = [

{ path: '', redirectTo: '/log-in', pathMatch: 'full' },

{ path: 'log-in', component: SigninComponent },

{ path: 'sign-up', component: SignupComponent },

{ path: 'user-profile/:id', component: UserProfileComponent, canActivate: [AuthGuard] }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

TypeScript

**Implement Reactive Forms**

Import ReactiveFormsModule and FormsModule in app.module.ts file and also declare in **imports: […]** array. [Check out more about Reactive forms in Angular](https://www.positronx.io/angular-8-express-file-upload-tutorial-with-reactive-forms/)

import { ReactiveFormsModule, FormsModule } from '@angular/forms';

@NgModule({

imports: [

ReactiveFormsModule,

FormsModule

],

})

export class AppModule { }

TypeScript

**Implementing User Registration in MEAN Stack App**

Now, implement user registration in MEAN stack auth app using Node API. Go to **components/signup.component.ts** file and add the following code.

import { Component, OnInit } from '@angular/core';

import { FormBuilder, FormGroup } from '@angular/forms';

import { AuthService } from './../../shared/auth.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-signup',

templateUrl: './signup.component.html',

styleUrls: ['./signup.component.scss'],

})

export class SignupComponent implements OnInit {

signupForm: FormGroup;

constructor(

public fb: FormBuilder,

public authService: AuthService,

public router: Router

) {

this.signupForm = this.fb.group({

name: [''],

email: [''],

mobile: [''],

password: [''],

});

}

ngOnInit() {}

registerUser() {

this.authService.signUp(this.signupForm.value).subscribe((res) => {

if (res.result) {

this.signupForm.reset();

this.router.navigate(['log-in']);

}

});

}

}

TypeScript

Go to **components/signup.component.html** file and add the following code inside of it.

<div class="auth-wrapper">

<form

class="form-signin"

[formGroup]="signupForm"

(ngSubmit)="registerUser()"

>

<h3 class="h3 mb-3 font-weight-normal text-center">Please sign up</h3>

<div class="form-group">

<label>Name</label>

<input

type="text"

class="form-control"

formControlName="name"

placeholder="Enter name"

required

/>

</div>

<div class="form-group">

<label>Email address</label>

<input

type="email"

class="form-control"

formControlName="email"

placeholder="Enter email"

required

/>

</div>

<div class="form-group">

<label>Password</label>

<input

type="password"

class="form-control"

formControlName="password"

placeholder="Password"

required

/>

</div>

<button type="submit" class="btn btn-block btn-primary">Sign up</button>

</form>

</div>

Markup

Call the **signUp()** method to register the user via the **registerUser()** method. On successful user registration redirect user to the log-in page.



**Handling MEAN Stack Login with Angular**

In this step, we will implement MEAN stack login in an Angular app. Go to **components/signin.component.ts** file and add the following code.

import { Component, OnInit } from '@angular/core';

import { FormBuilder, FormGroup } from '@angular/forms';

import { AuthService } from './../../shared/auth.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-signin',

templateUrl: './signin.component.html',

styleUrls: ['./signin.component.scss'],

})

export class SigninComponent implements OnInit {

signinForm: FormGroup;

constructor(

public fb: FormBuilder,

public authService: AuthService,

public router: Router

) {

this.signinForm = this.fb.group({

email: [''],

password: [''],

});

}

ngOnInit() {}

loginUser() {

this.authService.signIn(this.signinForm.value);

}

}

TypeScript

Head over to **components/signin.component.html** file and add the following code inside of it.

<div class="auth-wrapper">

<form class="form-signin" [formGroup]="signinForm" (ngSubmit)="loginUser()">

<h3 class="h3 mb-3 font-weight-normal text-center">Please sign in</h3>

<div class="form-group">

<label>Email</label>

<input

type="email"

class="form-control"

formControlName="email"

placeholder="Enter email"

required

/>

</div>

<div class="form-group">

<label>Password</label>

<input

type="password"

class="form-control"

formControlName="password"

placeholder="Password"

/>

</div>

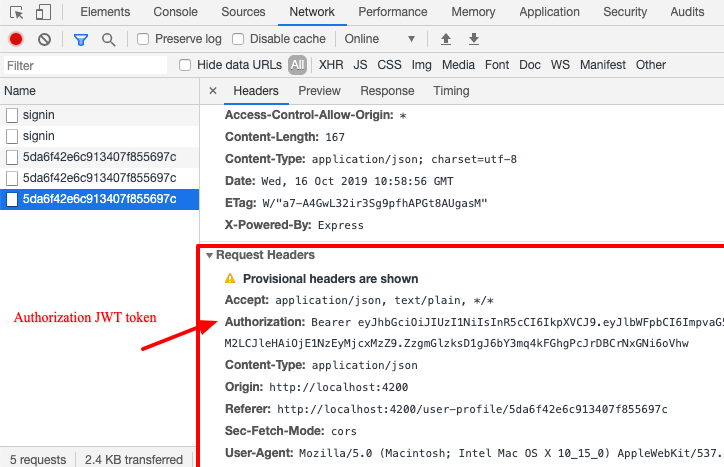
<button type="submit" class="btn btn-block btn-primary">Sign in</button>

</form>

</div>

Markup

Enter the user email and password, we are setting up Authorization: Bearer token in the header when the user successfully logged-in.



**Fetch User Profile in Angular Auth App**

Now, we will fetch the user data when the user is successfully logged in. In **server/ /middlewares/auth.js** file we have set the jwt.verify() method. This method checks the API request and does not render the user data if found invalid token or JWT secret.

For example try to access the /user-profile/\_id Angular URL without providing the invalid token. You will find out that server doesn’t render the user data.

Get into the **components/user-profile.component.ts** file and include the following code inside of it.

import { Component, OnInit } from '@angular/core';

import { ActivatedRoute } from '@angular/router';

import { AuthService } from './../../shared/auth.service';

@Component({

selector: 'app-user-profile',

templateUrl: './user-profile.component.html',

styleUrls: ['./user-profile.component.scss'],

})

export class UserProfileComponent implements OnInit {

currentUser: Object = {};

constructor(

public authService: AuthService,

private actRoute: ActivatedRoute

) {

let id = this.actRoute.snapshot.paramMap.get('id');

this.authService.getUserProfile(id).subscribe((res) => {

this.currentUser = res.msg;

});

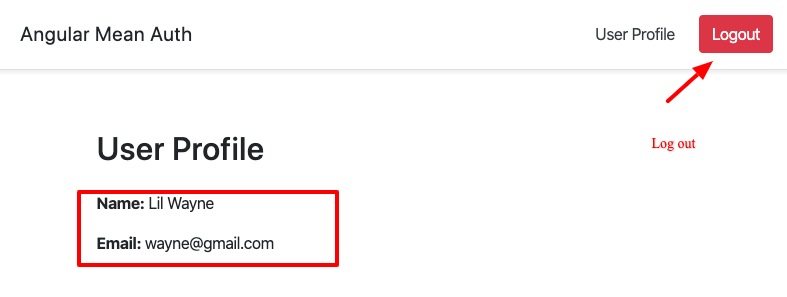
}

ngOnInit() {}

}

TypeScript

Get into the **components/user-profile.component.html** file and include the following code inside of it.



**Adding Logout in MEAN App**

In this step, we will add the logout, hiding and showing nav items in our MEAN stack user authentication app.

Go to **app/app.component.ts** file and add the following code inside of it.

import { Component } from '@angular/core';

import { AuthService } from './shared/auth.service';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.scss']

})

export class AppComponent {

constructor(public authService: AuthService) { }

logout() {

this.authService.doLogout()

}

}

TypeScript

Go to **app/app.component.html** file and add the following code inside of it.

<div

class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-white border-bottom shadow-sm fixed-top">

<h5 class="my-0 mr-md-auto font-weight-normal">Angular Mean Auth</h5>

<nav class="my-2 my-md-0 mr-md-3">

<a \*ngIf="this.authService.isLoggedIn" class="p-2 text-dark">User Profile</a>

<a \*ngIf="!this.authService.isLoggedIn" class="p-2 text-dark" routerLinkActive="active" routerLink="/log-in">Sign

in</a>

</nav>

<a \*ngIf="!this.authService.isLoggedIn" class="btn btn-outline-primary" routerLinkActive="active"

routerLinkActive="active" routerLink="/sign-up">Sign up</a>

<button (click)="logout()" \*ngIf="this.authService.isLoggedIn" type="button" class="btn btn-danger">Logout</button>

</div>

<router-outlet></router-outlet>

For reference <https://github.com/SinghDigamber/angular-meanstack-authentication>